Verifying the Conformance of a Driver Implementation to the VirtIO Specification

1st Matias Vara Larsen
Huawei Research Center
Grenoble, France
matias.vara.larsen@huawei.com

***Abstract*—VIRTIO is a specification that enables developers to base on a common interface to implement devices and drivers for virtual environments. This paper proposes the verification and analysis of the VIRTIO specification by using the Clock Constraint Specification Language (CCSL) [1]. In our proof-of-concept approach, a verification engineer translates requirements into a CCSL specification. Then, the tool TIMESQUARE [2] is used to detect inconsistencies with a implementation but also to understand what the specification enables. This paper aims to present the approach and to have face-to-face discussions and debate about the benefits, drawbacks and trade-offs.**

***Index Terms*—kernel, virtio, conformance, verification, formal**

I. INTRODUCTION

VIRTIO[1] is an specification for virtual environments that allows the Operating System (OS) to base on a simple mechanism to communicate with virtual devices. The use of this specification in emerging domains like automotive or cloud makes the specification in continuous development and review. Each *Virtual Machine Monitor* (VMM) relies on a different implementation of the VIRTIO devices, *e.g.*, QEMU[2], Rust-vmm[3] or hardware[4]. There are also different implementations of the virtio-drivers depending on the OS, *e.g.*, Linux, Windows or baremetal. The misinterpretation of the specification may lead to implementations that work well in some environments whereas they bug in others. In some cases, these unexpected behaviors are simply due to under-specified requirements.

To narrow the gap between specification and implementation, this paper proposes a proof-of-concept approach in which a *Verification Engineer* translates requirements by using the Clock Constraint Specification Language (CCSL) [1] to reach the following goals:

- understand what the specification enables
- validate the conformance of an implementation

We illustrate our approach by encoding the requirements regarding the initialization of virtio-devices. We use TimeSquare[5] [2] to understand what the specification allows and to verify a trace generated from the VIRTIO Linux driver.

Section II describes the approach. Section III uses this approach to analysis the initialization of virtio-devices. Section IV concludes this paper.

[1]https://docs.oasis-open.org/virtio/virtio/v1.1/cs01/virtio-v1.1-cs01.html
[2]https://www.qemu.org/
[3]https://github.com/rust-vmm
[4]https://lwn.net/Articles/805235/
[5]http://timesquare.inria.fr/

II. DESCRIPTION OF THE APPROACH

Figure 1 illustrates the workflow of a verification engineer that first translates requirements into a CCSL specification to later validate an implementation (in blue in Figure 1) and/or analyses the current specification (in green in Figure 1). The verification engineer can then use this analysis to refine the current specification (in red in Figure 1) or modify the current implementation (in orange in Figure 1).

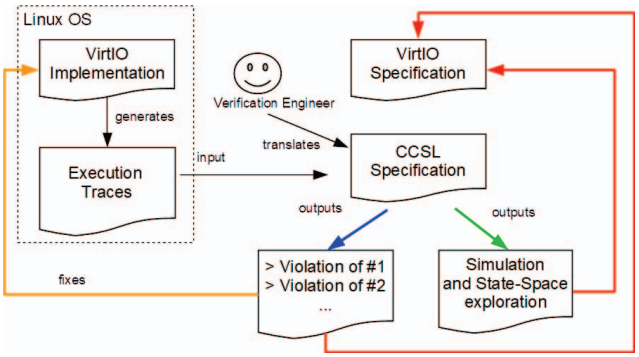

Fig. 1. Approach to Validate the VIRTIO specification

A verification engineer expresses requirements in term of events and partial relationships between them. An event is something that may occur like an interruption or a function call. A timed relationship specifies how the occurrences of events are related, *e.g.*, occur at the same time, one causes the second. In CCSL, clocks represent events, or more precisely the instants of a clock represent the CCSL occurrences of an event. A verification engineer may decide to use only the relevant events regarding the desired specification. These events can be conceptual and later on associated with a model element or combination of model elements. Relations between events are translated into *relationships*. In CCSL, relationships specify how the instances of clocks are related, *e.g.*, precedes, coincides, causality.

The resulting CCSL specification can be used to refine the requirements and/or validate an implementation. To do this, the verification engineer relies on the tool name TIMESQUARE that provides analysis features. Given a CCSL specification, TIMESQUARE finds a trace that conforms to the specification. This could result in no solution which means that the system is over-specified and the requirements shall be relaxed. When

the system is under-specified, the number of solutions may be infinite so the verification engineer may need to refine the specification to match the expected behavior. When the number of solutions is finite, TIMESQUARE allows a verification engineer to visualize the possible solutions by using an state-space diagram. This is an automata representation of the CCSL constraints. In the automaton, transitions are labeled with the events that must occur to reach such an state.

To validate an implementation, the verification engineer provides an execution trace of the implementation. In this paper, we focus on the traces generated from *ftrace*[6], which is the mechanism used by the Linux kernel to inform user about events that happens during kernel execution. These events can be function invocation or defined by user. The verification engineer feeds TIMESQUARE with the execution traces and the CCSL specification. TIMESQUARE informs when a trace violates a requirement. The verification engineer can either correct the implementation (in orange in Figure 1) or relax the requirements (in red in Figure 1).

III. USE CASE: VIRTIO DEVICE INITIALIZATION

The VIRTIO specification requires that virtio-drivers follow this sequence to initialize a device:

1) Set the *DRIVER* status bit. The driver informs the device that knows how to drive the device.
2) Sets the *FEATURES_OK* status bit.
3) Set the *DRIVER_OK* status bit. At this point the device is live.

From the specification, we identify three interesting events: *DRIVER*, *FEATURES_OK* and *DRIVER_OK*. We represent each of these events as CCSL clocks (see lines from 2 to 4 at Listing 1).

Listing 1. CCSL specification for VirtIO Device Status Register

```
ClockConstraintSystem VirtIOStatusRegister {
Clock Driver
Clock FeatureOK
Clock DriverOK
Relation DriverPrecedesFeatureOK [Precedes](LeftClock->
    Driver, RightClock->FeaturesOK)
Relation FeatureOKPrecedesDriverOK [Precedes](LeftClock->
    FeaturesOK, RightClock->DriverOK)
}
```

The specification requires that the driver first sets the *DRIVER* status bit and then sets the *FEATURES_OK* status bit. We express this partial relationship between these events by using the CCSL relationship *Precedes* between the clocks that represent such events (see line 5 at Listing 1). Similarly, we specify a precedence relationship between the events *FEATURES_OK* and *DRIVER_OK* (see line 6 at Listing 1). The specification results in two CCSL relationships named *DriverPrecedesFeatureOK* and *FeaturesOKPrecedesDriverOK*. By using TIMESQUARE, we generate the state-space diagram of the CCSL specification from Listing 1 (see Figure 2). We identify that the specification allows four possible states.

Listing 2. Ftrace output of the loading of virtio-ballon driver

```
insmod-221   [000] .N..    99.795755: Driver
```

[6]https://www.kernel.org/doc/Documentation/trace/ftrace.txt

Fig. 2. Automaton representation of the CCSL specification

```
insmod-221   [000] .N..    99.796669: FeaturesOK
insmod-221   [000] ....    99.806811: DriverOK
```

We use the CCSL specification to validate the implementation of the virtio-driver in the Linux kernel. Listing 2 shows the output of ftrace when the virtio-device is initialized. The fourth column informs the timestamp whereas the fifth column informs the name of the event. By importing the generated trace into TIMESQUARE, we validate that the trace conforms the specification (see Figure 3). Figure 4 shows a trace that violates

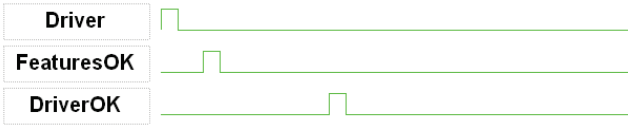

Fig. 3. VCD trace of the ftrace output

the specification. In this case, the *DRIVER_OK* occurs before the *FEATURES_OK*, which is forbidden by the specification. TIMESQUARE informs the violation by asserting the relation that has been violated, which is in this case the relationship *FeatureOKPrecedesDriverOK*.

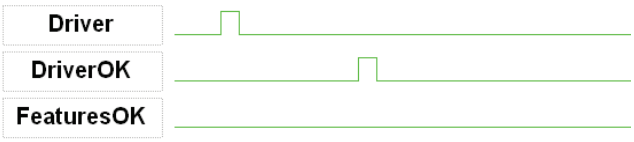

Fig. 4. VCD trace that violates the VIRTIO specification

IV. CONCLUSION

This paper has proposed a proof-of-concept approach to allow the validation of the requirements to initialize a virtio-device. We have shown how the formal specification of the requirements allows a verification engineer not only to understand what the specification allows but also to validate an existing implementation. Future work shall extend the analysis to the overall VIRTIO specification, cover more complex use-cases and understand how the specification could be revised as a result of the verification analysis. We look forward to get the VIRTIO community familiar with this approach and get feedback to improve the tooling.

REFERENCES

[1] Charles André. Syntax and Semantics of the Clock Constraint Specification Language (CCSL). [Research Report] RR-6925, INRIA. 2009, pp.37.

[2] Julien Deantoni, Frédéric Mallet. TimeSquare: Treat your Models with Logical Time. TOOLS - 50th International Conference on Objects, Models, Components, Patterns - 2012, Czech Technical University in Prague, in co-operation with ETH Zurich, May 2012, Prague, Czech Republic. pp.34-41.